### API Guide for Alvision Count Mobile Application

### Authentication Endpoints:

1. **POST** /auth/jwt/login
   * **Description**: Authenticates a user using their username and password, returning a JSON Web Token (JWT) for session management. The JWT token is required for accessing protected routes.
   * **Request Body**:
     + username (str): The user's username.
     + password (str): The user's password.
   * **Response**: A JWT token to be used for authenticating future requests.
2. **POST** /auth/register
   * **Description**: Registers a new user by creating a user account with the provided email and password.
   * **Request Body**:
     + email (str): The user's email address.
     + password (str): The user's password.
   * **Response**: The details of the newly created user.
3. **POST** /auth/request-password-reset
   * **Description**: Initiates the password reset process by sending a password reset token to the user's email.
   * **Request Body**:
     + email (str): The email address of the user requesting the password reset.
   * **Response**: A confirmation message indicating that the password reset token has been sent.
4. **POST** /auth/reset-password
   * **Description**: Resets the user's password using the provided token and new password.
   * **Request Body**:
     + token (str): The password reset token.
     + password (str): The new password.
   * **Response**: A confirmation message indicating that the password has been successfully reset.
5. **POST** /auth/verify
   * **Description**: Verifies the user's email address using the provided verification token.
   * **Request Body**:
     + token (str): The email verification token.
   * **Response**: A confirmation message indicating that the email has been successfully verified.
6. **GET** /users/me
   * **Description**: Retrieves the details of the currently authenticated user.
   * **Response**: The details of the authenticated user.

### User-Related Endpoints:

1. **GET** /authenticated-route
   * **Description**: An example route that requires authentication. Returns a greeting message for the authenticated user.
   * **Response**: A message greeting the authenticated user by their email.
2. **POST** /associate-user
   * **Description**: Associates a user with an admin, allowing the admin to manage the user's counts and data.
   * **Request Body**:
     + user\_email (str): The email address of the user to be associated with the admin.
   * **Response**: A confirmation message indicating that the user has been successfully associated with the admin.
3. **PATCH** /manual-count
   * **Description**: Manually updates the count of objects detected in the last processed image for the current user.
   * **Request Body**:
     + increment (int): The number by which to increment the object count.
     + processed\_image\_url (str): The URL of the processed image.
   * **Response**: The updated count record with the new count and the processed image URL.
4. **GET** /user-counts
   * **Description**: Retrieves all count records for the currently authenticated user.
   * **Response**: A list of all count records for the user.
5. **GET** /user-counts/{date}
   * **Description**: Retrieves count records for the currently authenticated user filtered by the specified date.
   * **Path Parameter**:
     + date (date): The date for which to retrieve count records.
   * **Response**: A list of count records for the specified date.

### Image Processing Endpoints:

1. **POST** /count
   * **Description**: Counts objects in an image using circle detection. Accepts a base64-encoded image, processes it, and returns the count of detected objects along with the URL of the processed image.
   * **Request Body**:
     + base64\_image (str): The base64-encoded image to be processed.
   * **Response**: The count of detected objects and the URL of the processed image.
2. **POST** /count-with-yolo
   * **Description**: Counts objects in an image using YOLO (You Only Look Once) detection. Accepts a base64-encoded image, processes it, and returns the count of detected objects along with the URL of the processed image.
   * **Request Body**:
     + base64\_image (str): The base64-encoded image to be processed.
   * **Response**: The count of detected objects and the URL of the processed image.

### Payment Endpoints:

1. **GET** /payment
   * **Description**: Renders the payment page for user subscriptions. Accepts the subscription type (monthly or yearly) and returns an HTML page for the payment process.
   * **Query Parameters**:
     + subscription\_type (str): The type of subscription, either 'monthly' or 'yearly'.
   * **Response**: An HTML page for processing the payment.
2. **POST** /payment/success
   * **Description**: Handles the successful payment process. Updates the user's subscription details based on the provided information.
   * **Request Body**:
     + email (str): The email address of the user.
     + order\_id (str): The order ID from the payment gateway.
     + subscription\_type (str): The type of subscription, either 'monthly' or 'yearly'.
   * **Response**: A confirmation message and updated user subscription details.
3. **GET** /subscription-check
   * **Description**: Checks the subscription status of the currently authenticated user. Returns the subscription details including status, type, start date, end date, and days remaining.
   * **Response**: The subscription details of the user.

### Utility Endpoints:

1. **GET** /logs
   * **Description**: Retrieves the application logs.
   * **Response**: The log file containing application logs.
2. **GET** /no-of-requests
   * **Description**: Gets the total number of requests made on a specific date across all users.
   * **Query Parameters**:
     + date (date): The date for which to retrieve the total number of requests.
   * **Response**: The total count of requests for the specified date.

### Health Check Endpoint:

1. **GET** /
   * **Description**: Health check endpoint for the application. Returns a status message indicating the application is running.
   * **Response**: A status message indicating the application is running.

### Middleware and Startup:

* **Middleware**: Logs each request and its response status code. The middleware logs the method and URL of the request, and the status code of the response. Additionally, it logs request stats using the system logger.
* **Startup Event**: Initializes the Beanie ODM with the MongoDB database and the User document model. This is performed during the application startup to ensure the database and models are ready for use.

### Models and Schemas:

* **CountRequest**: A schema for image count requests, containing a base64-encoded image.
* **UserCreate**, **UserRead**, **UserUpdate**: Schemas for creating, reading, and updating user details.
* **AWSConfig**: Manages AWS S3 configuration and image upload.
* **DetectCircle**, **count\_objects\_with\_yolo**: Modules for processing images and detecting objects.

### Notes:

* **Static Files**: The application serves static files from the ../static directory.
* **Templates**: The application uses Jinja2 templates located in the ../templates directory for rendering HTML responses.
* **Logging**: The application is configured to log request details and errors to help with debugging and monitoring.